Presampled Visibility for Ambient Occlusion

Athanasios Gaitatzes
University of Cyprus
75 Kallipoleos St.
P.O.Box.20537
Cyprus (CY-1678), Nicosia
gaitat@yahoo.com

Yiorgos Chrysanthou
University of Cyprus
75 Kallipoleos St.
P.O.Box.20537
Cyprus (CY-1678), Nicosia
yiorgos@cs.ucy.ac.cy

Georgios Papaioannou
Athens University of Economics and Business
76 Patission St.
Greece (10434), Athens
gepap@aueb.gr

ABSTRACT

We present a novel method to accelerate the computation of the visibility function of the lighting equation, in dynamic scenes composed of rigid, non-penetrating objects. The main idea of the technique is to pre-compute for each object in the scene its associated four-dimensional field that describes the visibility in each direction for all positional samples on a sphere around the object, we call this a displacement field. We are able to speed up the calculation of algorithms that trace visibility rays to near real time frame rates. The storage requirements of the technique, amounts from one byte to one bit per ray direction making it particularly attractive to scenes with multiple instances of the same object, as the same cached data can be reused, regardless of the geometric transformation applied to each instance. We suggest an acceleration technique and identify the sampling method that gives the best results based on experimentation.

Keywords

indirect lighting, pre-computed visibility, uniform distribution, hemisphere, queries, query-point, tracing rays.

1. INTRODUCTION

Ray based solutions to the rendering problem have been popular for over two decades now. An enormous amount of work has been done by researchers in order to accelerate the tracing of rays, especially through the use of spatial acceleration structures. However, such methods typically have a non-constant cost for ray-intersections. We propose an acceleration method for speeding up the visibility term of ray casting and apply the method to the approximation of the secondary diffuse illumination, namely the ambient occlusion.

Ambient occlusion is defined as the attenuation of ambient light due to the occlusion of nearby geometry. It is a technique that approximates the effect of indirect global illumination and does not yet try to simulate the interplay of incident and reflected light. In Ambient occlusion the indirect component can be computed as:

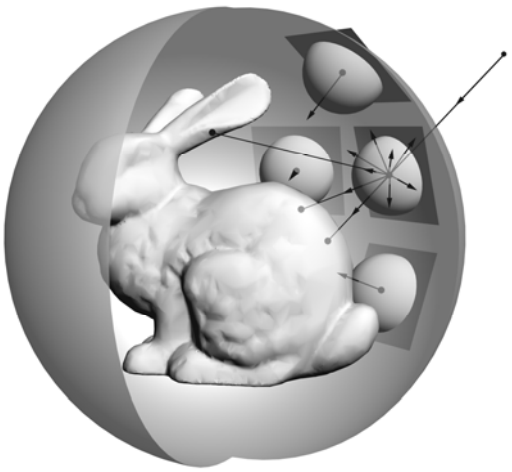


Figure 1: A hemisphere of rays emanating from the bounding sphere towards the object is precomputed for a large number of sample points on the sphere.

$$A(x,\vec{n}) = \frac{1}{\pi}\int_{\Omega} V(x,\vec{\omega}_o)(\vec{\omega}_o \cdot \vec{n})\, d\vec{\omega}_o$$

where $V(x,\vec{\omega}_o)$ is an empirical function that maps distance from surface point x to the closest surface along direction $\vec{\omega}_o$ to visibility values between 0 (no visibility) and 1.

```
generate bounding sphere sample points
generate samples of hemisphere of rays
for all bounding sphere sample points (u, v) do
  align hemisphere of rays to normal at (u, v)
  for all rays (φ, θ) do
    if ray intersects the object then
      normalize the distance (divide by 2 * R)
      record distance in displacement map
    else
      record distance in displacement map as 2 * R
  end
end
```

Algorithm 1: Pseudo code of basic algorithm for displacement fields computation at preprocessing time.

By tracing rays outward from a given surface point x over the hemisphere around the normal $\vec{n}$, ambient occlusion measures the amount that a point is obscured from light. This average occlusion factor is used to simulate soft-shadowing.

The method proposed in this paper uses a discretization approach. It accelerates the ray-object intersection test and in turn the computations of the visibility function of the lighting equation, by separating the task in two subtasks. First, at pre-processing time, we construct the displacement maps (Figure 1). These store the intersection distances of a hemisphere of rays originating from sample points on the bounding sphere of an object and directed towards the model itself. We construct one map for each sample point (Algorithm 1). Then, at run time, when a ray from the environment towards an object intersects its bounding sphere, we perform a simple ray-sphere intersection test and recover from the pre-computed maps the rest of the distance of the incoming ray at the given angle.

The advantage of our method is that the bulk of the computation is moved to a pre-processing stage. The results are stored in compact grayscale textures (one byte per ray direction), providing for each object a constant size of additional information independent of the complexity of the original model. Then the real time algorithm performs a simple intersection test and a constant-time map lookup as in Algorithm 2.

We show that, in applications such as ambient occlusion, maps that use 1-byte of storage per ray give almost the same result as maps that use 4-bytes of storage space. If the model changes level of detail the same maps can still be used. In addition, the displacement maps contain information that is transformation invariant. As such, no additional information has to be computed when the rigid object moves in the environment. For dynamic scenes with

```
generate hemisphere of ray samples
for each occlusion receiver object do
  for all points x on the occlusion receiver surface do
    for all emanating rays do
      if ray intersects bound sphere of occluder obj.
        discretize intersection point (u, v)
        discretize ray (φ, θ)
        access distance in displacement map
      end
      use distance for occlusion approximation
    end
    compute occlusion at x
  end
end
```

Algorithm 2: Pseudo code of basic algorithm for ambient occlusion rendering using displacement fields during real time processing.

rigidly moving objects, displacement fields accelerate the computation of the approximation of the indirect lighting term of the rendering equation to real-time frame rates as well as the computation of collision detection algorithms and ray casting.

In Section 2 we give an overview of the previous work, followed by a description of our method in greater detail in Section 3. In Section 4 we discuss our results in one application area, that of computing secondary diffuse illumination (termed ambient occlusion).

2. PREVIOUS WORK

We distinguish the previous work in three different areas: ray tracing acceleration algorithms, ambient occlusion computation, and various field computations around an object for accelerating different types of algorithms.

Ray Tracing Algorithms

Traditionally ray-scene intersection is accelerated through the use of hierarchical data structures. Bounding Volume Hierarchies [Gol87a] [Rub80a] [Cla76a], Voxel Grids [Sny87a] [Fuj86a], Hierarchical Grids [Kli97a] [Caz95a] [Jev89a], Octrees [Gla84a], Binary Space Partitioning Trees [Sun92a], kd-Trees [Hav02a] [Nay93a] [Mac90a] [Arv88b] [Jan86a] are just a few.

Recently, a new set of algorithms have been developed for interactive ray tracing and ray tracing of dynamic scenes. The work of Wald et al. demonstrates real time ray tracing for small scenes using in-expensive off-the-shelf PCs with SIMD floating point extensions [Wal01a] [Wal01b] and for larger scenes on shared memory multiprocessor machines by Parker et al. [Par98a] and on PCs using

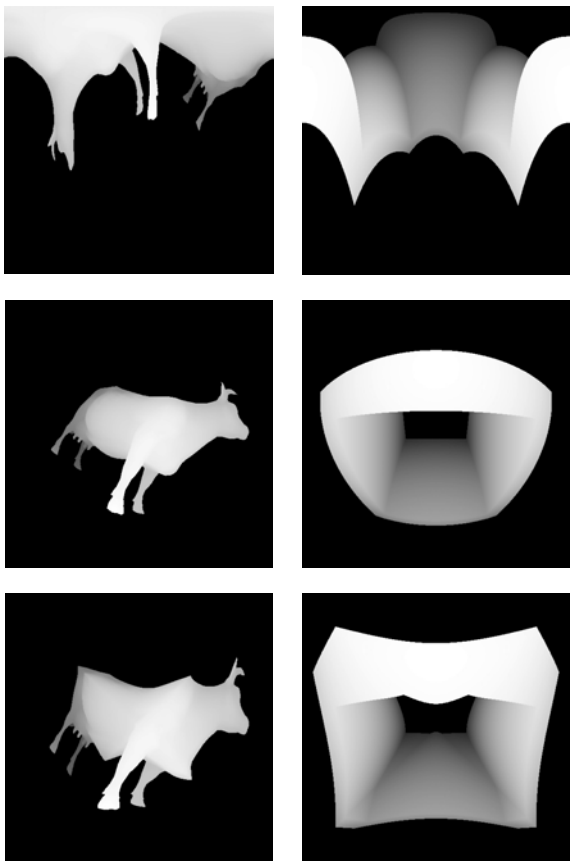


Figure 2: 512x512 displacement maps of a model of a cow and a cube with a hole in it. (top row) Using uniform Sampling of rays, (middle row) Rejection Sampling, (bottom row) Concentric Map Sampling. Different (θ, φ) to (s, t) mappings, produce different displacement maps.

a cluster architecture by Wald et al. [Wal01b] [Wal01a]. The main issue of these algorithms that accelerate spatially coherent rays, is that their speedup on secondary ray intersection tests is limited.

Ambient Occlusion

Ambient occlusion was first introduced by Zhukov and Iones et al. [Ion03a] [Zhu98a]. Their algorithm, depending on the size of the scene, could run in real time producing adequate results. For offline rendering, ambient occlusion is usually pre-computed at each vertex of the model, and stored either as vertex information or into a texture. For real-time rendering, recent work by Kontkanen et al. [Kon05a] suggests storing ambient occlusion as a field around moving objects, and projecting it onto the scene as the object moves. The interactions of multiple dynamically moving rigid objects can be combined in real-time. Zhou et al. [Zho05a] approximate the ambient occlusion by computing a field around an object that describes the shadowing effects of the model at points around it. The field is represented by Haar Wavelets or Spherical Harmonics making it more accurate than the method of Kontkanen et al. but also more expensive to calculate. Finally Malmer et al. [Mal05a] surround the object with a regular 3D grid, pre-computing ambient occlusion at the center of each grid cell with high memory costs for moderately complex scenes.

Field Computations around an Object

The work of Avneesh Sud et al. [Sud06b] [Sud04a] for computing the discretized 3D Euclidian distance to the surface of a primitive is used for speeding up interactive collision and distance queries types of algorithms. In our method, for the selected points around the object, we don't just compute the closest distance but rather the distance in a hemisphere of directions towards the object. In the work of Huang et al. [Hua06a] in a pre-computation stage the object is separated into convex segments each one surrounded by an oriented bounding box. The OBB is split into cells, each one recording a reference to the primitive that is intersected by a ray through this cell (traversal field). The multiple OBBs are needed in order to allow inter-reflections. Due to the fact that the number of OBBs and their corresponding traversal fields depends on the complexity of the original model, memory consumption may rise significantly.

3. DISPLACEMENT FIELDS

In this section we describe the general idea of displacement fields, while in section 4 we show their application for ambient occlusion.

Our method bears some similarity to the parameterization of Huang et al. [Hua06a] where each ray was described as a vector of the parametric incident location (u, v) on the bounding volume and its corresponding incoming direction (θ, φ). However, we introduce our novel displacement field encoding pre-computation where using a similar parameterization, we store the distance from the entry point on the bounding volume to the surface of the object. We further discuss the sampling techniques used and the storage requirements of our method along with the compression scheme.

Displacement Field Computation

The main idea of encoding displacement fields into maps is as follows (Algorithm 1). Consider a rigid object possibly moving through a scene. At a pre-processing step, from a discreet set of sample points on the bounding sphere, described as spherical coordinates (u, v), a hemisphere of rays is cast around the inward normal direction (Figure 1). For each ray (u, v, θ, φ), the closest distance between the

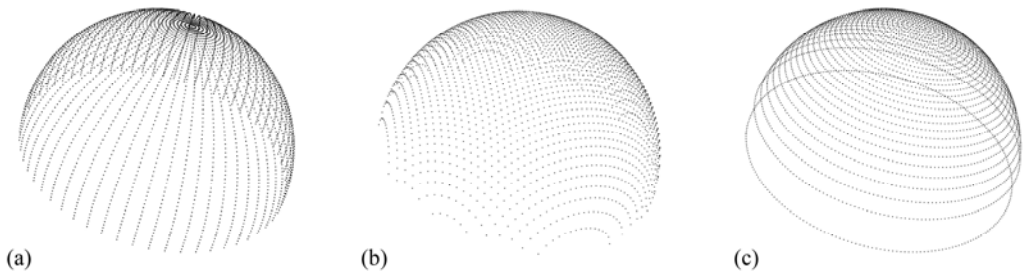


Figure 3: Sampling a hemisphere of rays. (a) Polar Mapping of rays, (b) Rejection Sampling, (c) Concentric Map Sampling.

bounding volume and the model surface is found and recorded as a compact integer value after being normalized by twice the sphere radius. Thus, for each sample point (u, v) a displacement grayscale map is obtained (Figure 2) that represents the distance traveled along the ray in the direction (θ, φ) before hitting the model surface. We define the displacement field of the object to be the collection of all displacement maps generated from all sample points on the bounding sphere of the object.

Displacement Field Indexing

During the real time part of the execution (Algorithm 2) an incident ray to the object, intersects its bounding sphere and the distance between the ray origin and the intersection point is recorded. The intersection point q is transformed into the object coordinate system: $\mathbf{q}' = \mathbf{M}^{-1} \cdot \mathbf{q}$, where $\mathbf{M}$ is the transformation matrix with respect to the reference frame of the ray. Depending on the sampling on the surface of the sphere (see Section 3.3), the inverse function is applied to $\mathbf{q}'$ in order to get the closest corresponding point (u, v) on the sphere for which we have a displacement map and therefore the index of the corresponding displacement map. Next we need to find the corresponding (θ, φ) of the incident ray. Depending on the ray sampling method (see section 3.4), the appropriate inverse function is applied to the ray, thus recovering the (θ, φ) values of the ray. We can now index into the displacement field for the given ray (u, v, θ, φ) and extract the distance information which is then added to the intersection distance above and this is our approximated distance value of the ray origin from the object's surface.

Selecting Samples around the Object

We need to sample entry points on the surface of the bounding volume of the object from where the rays originate in order to generate the displacement maps. The method selected must also have a quick inverse function that can convert an intersection point into the nearest sample. In addition it should distribute the samples over the bounding volume as evenly as possible.

A fairly straightforward choice are the spherical coordinates which have a fairly easy to compute inverse function. However, the samples in this method are concentrated more towards the poles of the sphere.

A common bounding shape that is used to sample the contained geometry is a axis-aligned bounding box (AABB). During the real-time simulation we would perform fast ray-box intersections. Special care though is needed as the AABBs are not transformation invariant and their oriented bounding boxes (OBB) counterparts require more operations.

As most sampling methods deal with sampling over a sphere, if the same methods were used to sample over a cube there would be a high concentration of samples near the vertices of the cube.

We opted for Slater's [Sla02a] method, which generates uniformly distributed points on a hemisphere using the triangle subdivision method. The same can be used to cover the full sphere as well. At the same time he suggests a constant time inverse function so, when an environment ray intersects the bounding sphere of the object, we can immediately associate this intersection point with one of the pre-generated displacement maps, in order to retrieve the angle and distance information.

Sampling a Hemisphere of Directions

There are several methods that deal with the uniform sampling of rays distributed over a hemisphere. The method selected must be able to uniquely discretize its samples so that they can be stored in the displacement maps. In addition there must exist an inverse function that converts the displacement map entries back into sample space.

One method is to use spherical coordinates where a direction in the hemisphere is given by two angles (φ, θ). But as can be seen in Figure 3a the rays generated are concentrated towards the cap of the

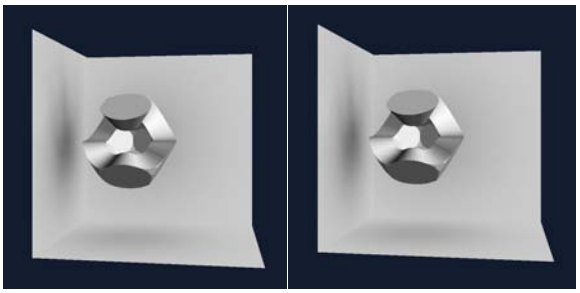


Figure 4: (left) 4 bytes per ray for storage, (right) 1 byte per ray for storage. There are no obvious visible differences, when the displacement fields are used for ambient occlusion.

hemisphere producing a good cosine term (close to 1.0) but they are not equally spaced.

In the rejection sampling method (Figure 3b) uniformly distributed points are selected inside a unit disk by selecting points inside the $[-1, 1]^2$ square and rejecting the points that fall outside the unit disk. Using Malley's method [Mal88a] the samples are projected on the disk up to the hemisphere above it, producing a cosine distribution of rays. Using this method, about 21.5% of the samples are rejected and so the corresponding space in the displacement map remains unused.

Shirley et al. [Shi97a] suggest a concentric map (Figure 3c) sampling method that maps samples in the square $[-1, 1]^2$ to the unit disk $\{(x, y) \mid x2 + y2 \leq 1\}$ by mapping concentric squares to concentric circles. The map preserves fractional area, it is bi-continuous and has low distortion. Combined with Malley's method where samples on the unit hemisphere have density proportional to the cosine term, it provides the best solution.

4. IMPLEMENTATION & RESULTS

We have implemented the displacement fields algorithm on an Intel Pentium 4 desktop PC running at 3.4 GHz with 1GB RAM and an nVIDIA Quadro FX 5500 graphics board, with 1GB Video RAM (mach. type 1) and an Intel dual Xeon running at 3.0 GHz with 4 GB RAM and the same graphics board (mach. type 2).

The implementation does not utilize the GPU for the indexing calculations. The method is a generic ray casting implementation, used in this case for ambient occlusion and as such can not be compared with other specialized GPU implementations.

Storage and Error Considerations

A 256x256 map stores the distance to the object for 65536 ray directions emanating from one sample. If that map was to store the values as floats it would require 262144 bytes of storage space while storing them as unsigned chars it would require 65536 bytes. In addition, if lossless compression is used (e.g. run length encoding) then on average less storage would be required. In application areas where integral calculations are performed over the samples or accuracy is not imperative, lossy compression could be used to further reduce the storage requirements. Given that it is essential to keep the storage requirements to a minimum we opted to use unsigned chars for storage as it was found in ambient occlusion approximations that there was little to no gain in visual quality from using floats as can be seen in Figure 4. If higher accuracy is desired one can consider storing more bytes per sample.

Results using the 8-bit maps

In our examples we used a large number of cast rays per vertex (256) and achieved interactive results that would otherwise be impossible (Figure 8). The complexity of the displacement field method is O (N_r) where N_r is the number of rays.

We have run several experiments in order to evaluate

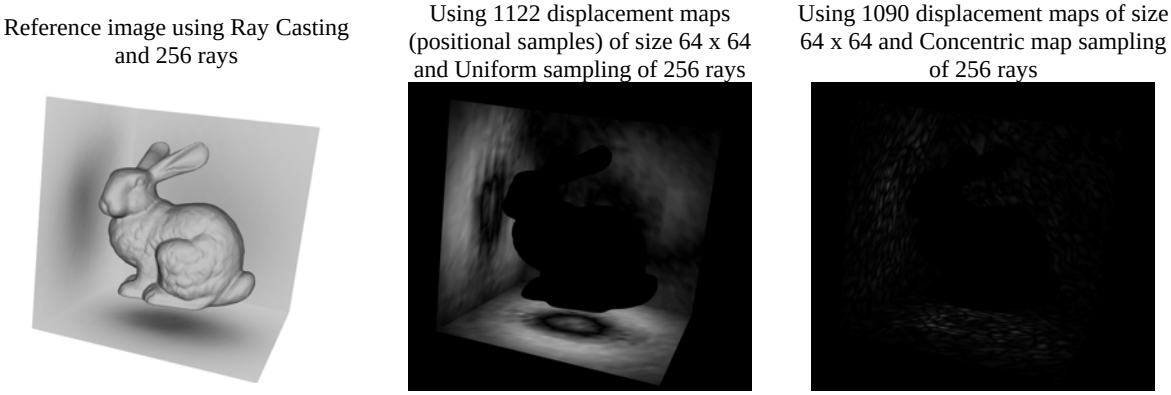


Figure 5: The image differences between the Reference image and the displacement map methods, show that using Concentric map sampling produces much better quality results as compared to Uniform sampling. Image differences are exaggerated by a factor of 5.

the method and establish which of the sampling method is the preferred.

As we can see in Figure 5 choosing a concentric map sampling distribution for the rays produces much better results than the uniform sampling of rays. As such, we opted to use this method in producing the rest of the results.

In Figure 6 we see images of the ambient occlusion solution produced using 4 different resolutions for the concentric map sampling for the ray directions and 3 resolutions for the positional samples on the bounding sphere around the object. Here, as expected, we see that cost of computing the displacement field is a function of the sampling resolutions while the cost of using it for ambient occlusion is almost independent of the object complexity. By looking at the root mean square (rms) error, we observe that it drops quickly as we increase the positional samples. In addition, we observe that as we increase the directional samples, the error does not decrease significantly. So a good compromise between memory use and accuracy would be to use the 4226 / 32x32 maps.

In Figure 8, we see the method applied to different types of models. We observe that the cost of using the displacement maps increases very little as we go to higher complexity models. The only exception is the multiple model case, where we have inter-object interactions. The bunny is a caster, the corner is a receiver and the other two objects are both casters and receivers. So the results are justified by the increase of rays cast by about 30 times.

Further Memory Optimization

When objects are further away from the viewer, the approximate ambient occlusion calculated previously, can be further optimized in terms of texture space required. Instead of storing into the map the distance between the bounding sphere and the object, we can store only the visibility of the

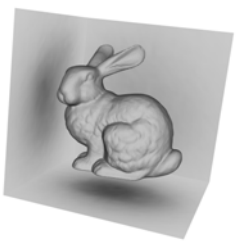
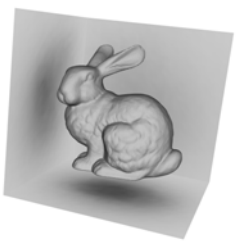
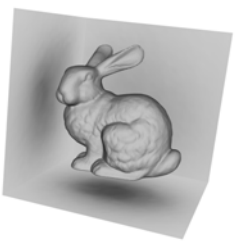
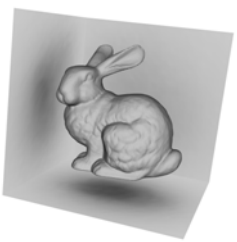
Displacement field positional samples	Displacement field directional samples: 32 x 32	64 x 64	128 x 128	256 x 256
290	 3.96 / 0.09462 / 2.8207	 14.60 / 0.09584 / 2.4118	 54.42 / 0.09809 / 2.3153	 213.75 / 0.10159 / 2.2914
1090	 14.06 / 0.09870 / 1.0653	 51.23 / 0.10219 / 1.0318	 193.25 / 0.10785 / 1.0294	 772 / 0.11516 / 1.02871
4226	 54.79 / 0.10826 / 0.6772	 204.27 / 0.11118 / 0.6315	 925.05 / 0.11504 / 0.6209	 3039.72 / 0.12190 / 0.6185

Figure 6: Cumulative table using 256 sample rays from each vertex of the tessellated corner (3x33x33) with a concentric map sampling distribution. The numbers under the images correspond to the pre-processing time, the run-time ambient occlusion computation in seconds (using mach. type 2) and the rms error compared to the Reference image of Figure 5.

geometry in the given ray direction. This is a binary value, thus the method saves about 87.5% in texture space. The distance used in this case is the average distance of the sample points towards the object in the direction of the normal at the given sample point. In Figure 7 we can see the comparable results.

5. CONCLUSIONS - FUTURE WORK

We have presented the displacement fields, a novel discretization of the visibility around an object. We have shown how it can be used for an interactive ambient occlusion approximation computation. It especially favors large model data sets, where we maintain a constant computation time, independent of the model complexity as shown in Figure 8. Our method is robust, has a relatively small memory footprint against comparable existing methods and the time required to generate the displacement maps depends only on the complexity of the occluder geometry. Furthermore, our algorithm can be applied to ray tracing calculations where exact ray hits are not critical, for example for secondary ray intersection tests, such as soft shadow rays.

The number and resolution of the displacement maps used in the displacement field can be adjusted depending on the required accuracy and available memory.

In addition we could try to use a smaller number of displacement maps around the sphere; just enough to cover the surface of the sphere with little map overlap. We would in effect be creating an environment map. But then, our approximation would suffer as the angle of approximation increases, especially if our objects' geometry possessed many folds and creases that would provide dramatic self-occlusion variations from different angles.

Figure 7: Using the 1 bit per direction optimization method with 4226 occlusion maps (positional samples) of size 64x64 and Concentric map sampling of 256 rays we get results which are comparable with the corresponding image from Figure 6 but slightly brighter (giving an rms error of 4.4030).

For future work, for objects that leave too much void space in their bounding sphere, a hierarchical scheme could be used such as a sphere tree [Bra02a]. This would result in tighter sphere placement and denser partial displacement maps at the expense of texture storage.

Furthermore, it is possible to map the displacement field indexing procedure to a shader program and stack the displacement maps into one 3D texture. Then the distance determination can be executed in the GPU with the added advantage of a trilinear interpolation of the distance for an arbitrary ray from

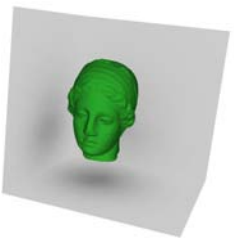
	Lemon Tree	Bunny	Igea	Multiple Objects
Model				
Triangles	26,300	39,000	67,200	142,300
Rays cast	836,352	836,352	836,352	26,444,800
Ray casting time	196.20 s	331.10 s	616.35 s	4,286.8 s
Pre-processing (4226 / 32x32)	99.54 s	54.79 s	243.76 s	334.6 s
AO calculation	0.240 s	0.228 s	0.204 s	4.692 s

Figure 8: The displacement field algorithm applied to several different types of models and their respective timings (using mach. type 1). In the above images we used 256 sample rays with a concentric map sampling distribution. The ambient occlusion computation is done using the 4226 / 32x32 maps.

distances measured at the discrete sample points (third 3D texture coordinate) and discrete directions (s, t plane texels).

6. REFERENCES

[Arv97a] Arvo, J., Kirk, D.: Fast ray tracing by ray classification. In Proc. SIGGRAPH '97 (1997).

[Arv88b] Arvo, J.: Linear time voxel walking for octrees. Ray Tracing News 12(1), (1988).

[Bra02a] Bradshaw, G., O'Sullivan, C.: Sphere-Tree Construction using Medial-Axis Approximation. In Proc. of the 2002 ACM Symposium on Computer Animation, SCA 2002, San Antonio, Texas.

[Caz95a] Cazals, F., Drettakis, G., Puech, C.: Filtering, Clustering and Hierarchy Construction: A new solution for ray tracing complex scenes. Computer Graphics Forum 14, 3 (1995), pp. 371–382.

[Cla76a] Clark, J. H.: Hierarchical geometric models for visible surface algorithms. Communications of the ACM 19, 10 (1976), pp. 547–554.

[Fuj86a] Fujimoto, A., Tanaka, T., Iwata, K.: Arts: Accelerated ray tracing system. In Proc. IEEE Computer Graphics and Applications 6, 4 (1986), pp. 16–26.

[Gla84a] Glassner, A.: Space subdivision for fast ray tracing. In Proc. IEEE Computer Graphics and Applications 4, 10 (1984), pp. 15–22.

[Gol87a] Goldsmith, J., Salmon, J.: Automatic creation of object hierarchies for ray tracing. In Proc. IEEE Computer Graphics and Applications 7, 5 (1987), pp. 14–20.

[Hav02a] Havran, V., Bittner, J.: On improving kd-trees for ray shooting. In Proc. of WSCG '02 Conference, (2002), pp. 209–17.

[Hua06a] Huang, P., Wang, W., Yang, G., Wu, E.: Traversal fields for ray tracing dynamic scenes. In ACM Symposium on Virtual Reality Software and Technology (VRST '06), Limassol Cyprus, November 1-3, 2006.

[Ion03a] Iones, A., Krupkin, A., Sbert, M., Zhukov, S.: Fast, realistic lighting for video games. In Proc. IEEE Computer Graphics and Applications 23, 3 (May 2003), pp. 54–64.

[Jan86a] Jansen, F. W.: Data structures for ray tracing. In L. R. A. Kessener, F. J. Peters, and M. L. P. Lierop (Eds.), Data Structures for Raster Graphics, Workshop Proceedings, pp. 57–73. New York: Springer–Verlag, 1986.

[Jev89a] Jevans, D., Wyvill, B.: Adaptive voxel subdivision for ray tracing. In Proc. Graphics Interface, (1989), pp. 164–172.

[Kli97a] Klimaszewski, K. S., Sederberg, T. W.: Faster ray tracing using adaptive grids. In IEEE Computer Graphics and Applications 17, 1 (1997), pp. 42–51.

[Kon05a] Kontkanen, J., Laine, S.: Ambient occlusion fields. In Proc. Interactive Symposium on 3D Graphics, (2005).

[Mac90a] MacDonald, J. D., Booth, K. S.: Heuristics for ray tracing using space subdivision. The Visual Computer 6, 3 (1990), pp. 153–66.

[Mal88a] Malley, T. J. V.: A shading method for computer generated images. In Master's Thesis, Computer Science Department, University of Utah, June (1988).

[Mal05a] Malmer, M., Malmer, F., Assarsson, U., Holzschuch, N.: Fast pre-computed ambient occlusion for proximity shadows. In Tech. Rep. RR-5779, INRIA, (2005).

[Nay93a] Naylor, B.: Constructing good partition trees. In Graphics Interface, (1993), pp. 181–191.

[Par99a] Parker, S., Martin, W., Sloan, P.-P., Shirley, P., Smits, B., Hansen, C.: Interactive ray tracing. In ACM Symposium on Interactive 3D Graphics, (1999), pp. 119–126.

[Rub80a] Rubin, S. M., Whitted, T.: A 3-dimensional representation for fast rendering of complex scenes. In Computer Graphics 14, 3 (1980), pp. 110–116.

[Shi97a] Shirley, P., Chiu, K.: A low distortion map between disk and square. In Journal of Graphics Tools 2, 3 (1997).

[Sla02a] Slater, M.: Constant time queries on uniformly distributed points on a hemisphere. In Journal of Graphic Tools 7, 1 (2002), pp. 33–44.

[Sny87a] Snyder, J. M., Barr, A. H.: Ray tracing complex models containing surface tessellations. In M. C. Stone (Ed.), Computer Graphics (SIGGRAPH '87 Proceedings), 21 (1987), pp. 119–128.

[Sud04a] Sud, A., Otaduy, M. A., Manocha, D.: DiFi: Fast 3D distance field computation using graphics hardware. In Computer Graphics Forum 23, 3 (2004), pp. 557–566.

[Sud06b] Sud, A., Govindaraju, N., Gayle, R., Manocha, D.: Interactive 3D distance field computation using linear factorization. In Proc. ACM Symposium on Interactive 3D Graphics and Games (I3D 2006).

[Sun92a] Sung, K., Shirley, P.: Ray tracing with the BSP tree. In D. Kirk (Ed.), Graphics Gems III, (1992), pp. 271-274. San Diego: Academic Press.

[Wal01a] Wald, I., Slusallek, P., Benthin, C.: Interactive distributed ray tracing of highly complex models. In Rendering Techniques 2001, 12th Eurographics Workshop on Rendering, (2001), pp. 277–288.

[Wal01b] Wald, I., Slusallek, P., Benthin, C., Wagner, M.: Interactive rendering with coherent ray tracing. In Computer Graphics Forum 20, 3 (2001), pp. 153–164.

[Zho05a] Zhou, K., Hu, Y., Lin, S., Guo, B., Shum, H.-Y.: Pre-computed shadow fields for dynamic scenes. In SIGGRAPH 2005, pp. 1196-1201.

[Zhu98a] Zhukov, S., Iones, A., Kronin, G.: An ambient light illumination model. In Rendering Techniques—Proc. Eurographics Rendering Workshop, Springer-Wien, 1998, pp. 45-55.